

PATENT
5500-91600
TT5403

"EXPRESS MAIL" MAILING LABEL
NUMBER EV 318246965 US
DATE OF DEPOSIT 11-3-03

I HEREBY CERTIFY THAT THIS PAPER OR
FEE IS BEING DEPOSITED WITH THE
UNITED STATES POSTAL SERVICE
"EXPRESS MAIL POST OFFICE TO
ADDRESSEE" SERVICE UNDER 37 C.F.R. §
1.10 ON THE DATE INDICATED ABOVE
AND IS ADDRESSED TO THE
COMMISSIONER FOR PATENTS, BOX
PATENT APPLICATION, WASHINGTON,
D.C. 20231


Derrick Brown

Instruction Cache Prefetch Based on Trace Cache Eviction

Gregory William Smaus

Mitchell Alsup

BACKGROUND OF THE INVENTION

Field of the Invention

5 **[0001]** This invention is related to the field of microprocessors, and more particularly, to microprocessors having trace caches.

Description of the Related Art

10 **[0002]** Instructions processed in a microprocessor are encoded as a sequence of ones and zeros. For some microprocessor architectures, instructions may be encoded with a fixed length, such as a certain number of bytes. For other architectures, such as the x86 architecture, the length of instructions may vary. The x86 microprocessor architecture specifies a variable length instruction set (i.e., an instruction set in which various
15 instructions are each specified by differing numbers of bytes). For example, the 80386 and later versions of x86 microprocessors employ between 1 and 15 bytes to specify a particular instruction. Instructions have an opcode, which may be 1-2 bytes, and additional bytes may be added to specify addressing modes, operands, and additional details regarding the instruction to be executed.

20 **[0003]** In some microprocessor architectures, each instruction may be decoded into one or more simpler operations prior to execution. Decoding an instruction may also involve accessing a register renaming map in order to determine the physical register to which each logical register in the instruction maps and/or to allocate a physical register to
25 store the result of the instruction.

30 **[0004]** In general, the bandwidth of the instruction fetch and decode portions of a microprocessor may determine whether the execution cores are fully utilized during each execution cycle. Accordingly, it is desirable to be able to provide enough bandwidth in the instruction fetch and decode portions of the microprocessor to kept the execution core as fully supplied with work as possible.

[0005] Typically, instructions are fetched from system memory into instruction cache in contiguous blocks. The instructions included in these blocks are stored in the instruction cache in compiled order. During program execution, instructions are often executed in a different order than compiled order. For example, when a branch is taken within the code, non-sequential instructions (in compiled order) may be executed sequentially. In such cases, the instructions following the taken branch cannot generally be fetched from the instruction cache during the same cycle as the branch instruction because they are stored in non-contiguous locations. To attempt to overcome this instruction fetch bandwidth limitation, a superscalar microprocessor may incorporate a trace cache.

[0006] Trace cache differs from instruction cache in that instructions stored in trace cache are typically stored in execution order as opposed to compiled order. Storing operations in execution order allows a trace containing one or more taken branch operations to be accessed during a single cycle from trace cache, whereas accessing the same sequence from instruction cache would require several cycles.

SUMMARY

[0007] Various embodiments of methods and systems for implementing a
5 microprocessor that fetches a group of instructions into instruction cache in response to a
corresponding trace being evicted from the trace cache are disclosed. In some
embodiments, a microprocessor may include an instruction cache, a trace cache, and a
prefetch unit. In response to a trace being evicted from trace cache, the prefetch unit may
fetch a line of instructions into instruction cache.

10 [0008] In some embodiments, the prefetch unit may fetch a plurality of lines of
instructions into instruction cache in response to a trace being evicted from the trace
cache. If an evicted trace includes one or more branch instructions, the prefetch unit may
fetch a line including the first instruction (in program order) of the evicted trace, as well
15 as one or more lines including the instructions following the predicted-taken branch
instructions included within the trace. For example, if an evicted trace includes two
predicted-taken branch instructions, the prefetch unit may respond to the eviction of the
trace by fetching three lines of instructions into instruction cache. The first line fetched
may include the first instruction of the evicted trace. The second line fetched may
20 include the first instruction following the first branch instruction of the evicted trace and
the third line fetched may include the first instruction following the second branch
instruction of the evicted trace.

[0009] In some embodiments, the prefetch unit may check certain conditions to
25 determine whether or not to fetch a line of instructions into instruction cache in response
to the eviction of a trace from trace cache. For example, when a given trace is evicted
from trace cache, statistics recording the frequency of use of the trace may be examined.
If the frequency of use for the evicted trace is above a threshold value, the prefetch unit
may fetch one or more lines of instructions corresponding to instruction included in the
30 evicted trace into the instruction cache. If the frequency of use for the evicted trace is

below the threshold value, the prefetch unit may inhibit the fetching of lines of instructions into instruction cache.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] A better understanding of the present invention can be obtained when the following detailed description is considered in conjunction with the following drawings,
5 in which:

[0011] FIG. 1 shows a microprocessor incorporating a trace cache, according to one embodiment.

10 [0012] FIG. 2 illustrates an exemplary trace cache entry, according to one embodiment.

[0013] FIG. 3 is a flowchart for a method for fetching instructions from an instruction cache in response to the eviction of traces from a trace cache, according to one
15 embodiment.

[0014] FIG. 4 shows one embodiment of a computer system.

[0015] FIG. 5 shows another embodiment of a computer system.
20

[0016] While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular
25 form disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the present invention as defined by the appended claims. Note, the headings are for organizational purposes only and are not meant to be used to limit or interpret the description or claims. Furthermore, note that the word "may" is used throughout this application in a permissive sense (i.e.,
30 having the potential to, being able to), not a mandatory sense (i.e., must). The term "include" and derivations thereof mean "including, but not limited to." The term

“connected” means “directly or indirectly connected,” and the term “coupled” means “directly or indirectly coupled.”

DETAILED DESCRIPTION OF EMBODIMENTS

[0017] FIG. 1 is a block diagram of logical components included in one embodiment of a microprocessor 100 that fetches a group of instructions into instruction cache 106 in response to a corresponding trace being evicted from the trace cache 160. Microprocessor 100 is configured to execute instructions stored in a system memory 200. Many of these instructions operate on data stored in system memory 200. Note that system memory 200 may be physically distributed throughout a computer system and may be accessed by one or more microprocessors 100. In some embodiments, the microprocessor 100 may be designed to be compatible with the x86 architecture. Note that microprocessor 100 may also include and/or be coupled to many other components in addition to those shown here. For example, additional levels of cache may be included (internal and/or external to microprocessor 100) between microprocessor 100 and system memory 200. Similarly, microprocessor 100 may include a memory controller configured to control system memory 200 in some embodiments. Additionally, the interconnections between logical components may vary between embodiments.

[0018] Microprocessor 100 may include an instruction cache 106 and a data cache 128. Microprocessor 100 may include a prefetch unit 108 coupled to the system memory 200. Prefetch unit 108 is a means for prefetching instruction code from the system memory 200 for storage within instruction cache 106. In one embodiment, prefetch unit 108 may be configured to burst code from the system memory 200 into instruction cache 106. Prefetch unit 108 may employ a variety of specific code prefetching techniques and algorithms. Prefetch unit 108 may also fetch instructions from instruction cache 106 and traces from trace cache 160 into dispatch unit 104. Instructions may be fetched from instruction cache 106 in response to a given instruction address missing in trace cache 160. Likewise, instructions may be fetched from system memory 200 in response to a given address missing in instruction cache 106. Prefetch unit 108 may also fetch instructions corresponding to a trace from instruction cache 106 in response to the trace being evicted from trace cache 160 as described in more detail below.

5 [0019] A dispatch unit 104 may be configured to receive instructions from instruction cache 106 and to receive decoded and/or partially decoded operations from trace cache 160. The dispatch unit 104 may include a decode unit 140 to decode instructions received from instruction cache 106. The dispatch unit 104 may also include a microcode unit for use when handling microcoded instructions.

10 [0020] The dispatch unit 104 is configured to dispatch operations to scheduler(s) 118. One or more schedulers 118 may be coupled to receive dispatched operations from dispatch unit 104 and to issue operations to one or more execution cores 124. Results generated by execution core(s) 124 may be output to a result bus 130. These results may be used as operand values for subsequently issued instructions and/or stored to register file 116. A retire queue 102 may be coupled to scheduler(s) 118 and dispatch unit 104. The retire queue may be configured to determine when each issued operation may be retired.

15 [0021] Instruction cache 106 may temporarily store instructions prior to their receipt by dispatch unit 104. Instruction code may be provided to instruction cache 106 by prefetching code from the system memory 200 through prefetch unit 108. Instruction cache 106 may be implemented in various configurations (e.g., set-associative, fully-associative, or direct-mapped).

20 [0022] Dispatch unit 104 may output signals including bit-encoded operations executable by the execution core(s) 124 as well as operand address information, immediate data and/or displacement data. Decode unit 140 may be used to decode certain instructions into one or more operations executable within execution core(s) 124. Simple instructions may correspond to a single operation. More complex instructions may correspond to multiple operations. Upon receiving an operation that involves the update of a register, the dispatch unit 104 may reserve a register location within register file 116 to store speculative register states (in an alternative embodiment, a reorder buffer may be used to store one or more speculative register states for each register). A register map may translate logical register names of source and destination operands to physical

register names in order to facilitate register renaming. Such a register map may track which registers within register file 116 are currently allocated and unallocated.

5 [0023] When operations are handled by dispatch unit 104, if a required operand is a register location, register address information may be routed to a register map or a reorder buffer. For example, in the x86 architecture, there are eight 32-bit logical registers (e.g., EAX, EBX, ECX, EDX, EBP, ESI, EDI and ESP). Physical register file 116 (or a reorder buffer) includes storage for results that change the contents of these logical registers, allowing out of order execution. A physical register in register file 116
10 may be allocated to store the result of each operation that modifies the contents of one of the logical registers. Therefore, at various points during execution of a particular program, register file 116 (or, in alternative embodiments, a reorder buffer) may have one or more registers that contain the speculatively executed contents of a given logical register.

15 [0024] A register map may assign a physical register to a particular logical register specified as a destination operand for an operation. Register file 116 may have one or more previously allocated physical registers assigned to a logical register specified as a source operand in a given operation. The register map may provide a tag for the physical
20 register most recently assigned to that logical register. This tag may be used to access the operand's data value in the register file 116 or to receive the data value via result forwarding on the result bus 130. If the operand corresponds to a memory location, the operand value may be provided on the result bus for result forwarding and/or storage in register file 116. Operand data values may be provided to execution core(s) 124 when
25 the operation is issued by one of the scheduler(s) 118. Note that in alternative embodiments, operand values may be provided to a corresponding scheduler 118 when an operation is dispatched (instead of being provided to a corresponding execution core 124 when the operation is issued).

30 [0025] The microprocessor 100 of FIG. 1 supports out of order execution. A retire queue 102 (or, alternatively, a reorder buffer) may keep track of the original program

sequence for register read and write operations, allow for speculative instruction execution and branch misprediction recovery, and facilitate precise exceptions. In many embodiments, retire queue 102 may function similarly to a reorder buffer. However, unlike a typical reorder buffer, retire queue 102 may not provide any data value storage.

5 In alternative embodiments, retire queue 102 may function more like a reorder buffer and also support register renaming by providing data value storage for speculative register states. In some embodiments, retire queue 102 may be implemented in a first-in-first-out configuration in which operations move to the “bottom” of the buffer as they are validated, thus making room for new entries at the “top” of the queue. As operations are
10 retired, retire queue 102 may deallocate registers in register file 116 that are no longer needed to store speculative register states and provide signals to a register map indicating which registers are currently free. By maintaining speculative register states within register file 116 (or, in alternative embodiments, within a reorder buffer) until the operations that generated those states are validated, the results of speculatively-executed
15 operations along a mispredicted path may be invalidated in the register file 116 if a branch prediction is incorrect.

[0026] Retire queue 102 may also provide signals identifying program traces to trace generator 170. Trace generator 170 may also be described as a fill unit. Trace generator
20 170 may store traces identified by retire queue 102 into trace cache 160. Each trace may include instructions included in several different basic blocks. A basic block may be a set of instructions that are all executed as a unit. For example, a group of instructions that begins just after a branch instruction and ends with another branch instruction may comprise a basic block. If the first instruction of the basic block is executed, then all
25 instructions included in the basic block may be executed, otherwise none of the instructions within the basic block may be executed. The traces stored into trace cache 160 may include several decoded or partially decoded instructions. As used herein, a “trace” is a group of instructions that are stored within a single trace cache entry in the trace cache 160. Trace generator 170 is a means for generating traces.

30

[0027] Prefetch unit 108 may fetch instructions from trace cache 160 into dispatch unit 104. In some embodiments in which instructions in trace cache are at least partially decoded, when operations are fetched from the trace cache (as opposed to when instructions are loaded from instruction cache 106), the decode unit 140 may be at least partially bypassed, resulting in a decreased number of dispatch cycles for the cached operations. Accordingly, the trace cache 160 may allow the dispatch unit 104 to amortize the time taken to partially (or fully) decode the cached operations in decode unit 140 over several execution iterations if traces are executed more than once.

10 [0028] The bit-encoded operations and immediate data provided at the outputs of dispatch unit 104 may be routed to one or more schedulers 118. Note that as used herein, a scheduler is a device that detects when operations are ready for execution and issues ready operations to one or more execution units. For example, a reservation station is a scheduler. Each scheduler 118 may be capable of holding operation information (e.g., bit
15 encoded execution bits as well as operand values, operand tags, and/or immediate data) for several pending operations awaiting issue to an execution core 124. In some embodiments, each scheduler 118 may not provide operand value storage. Instead, each scheduler may monitor issued operations and results available in register file 116 in order to determine when operand values will be available to be read by execution core(s) 124
20 (from register file 116 or result bus 130). In some embodiments, each scheduler 118 may be associated with a dedicated execution core 124. In other embodiments, a single scheduler 118 may issue operations to more than one of the execution core(s) 124.

[0029] Schedulers 118 may be provided to temporarily store operation information to be executed by the execution core(s) 124. As stated previously, each scheduler 118 may
25 store operation information for pending operations. Additionally, each scheduler may store operation information for operations that have already been executed but may still reissue. Operations are issued to execution core(s) 124 for execution in response to the values of any required operand(s) being made available in time for execution.
30 Accordingly, the order in which operations are executed may not be the same as the order of the original program instruction sequence.

[0030] In one embodiment, each of the execution core(s) 124 may include components configured to perform integer arithmetic operations of addition and subtraction, as well as shifts, rotates, logical operations, and branch operations. A floating point unit may also be included to accommodate floating point operations. One or more of the execution core(s) 124 may be configured to perform address generation for load and store memory operations.

[0031] The execution core(s) 124 may also provide information regarding the execution of conditional branch instructions to branch prediction unit 132. If information from the execution core 124 indicates that a branch prediction is incorrect, the branch prediction unit 132 may flush instructions subsequent to the mispredicted branch that have entered the instruction processing pipeline and redirect prefetch unit 108. The redirected prefetch unit 108 may then begin fetching the correct set of instructions from instruction cache 106, trace cache 160, and/or system memory 200. In such situations, the results of instructions in the original program sequence that occurred after the mispredicted branch instruction may be discarded.

[0032] Results produced by components within execution core(s) 124 may be output on the result bus 130 to the register file 116 if a register value is being updated. If the contents of a memory location are being changed, the results produced within execution core(s) 124 may be provided to data cache unit 128.

Trace Cache

[0033] Trace generator 170 may be configured to receive retired instructions from retire queue 102 and to store those retired operations within traces in trace cache 160. As mentioned above, instructions may be stored into traces within the trace cache 160 in execution order as opposed to program order. Note that in alternative embodiments, trace generator 170 may be coupled to the front-end of the microprocessor (e.g., before or after the dispatch unit) and configured to generate traces from instructions detected within the pipeline at that point within the microprocessor.

[0034] Under certain conditions, as described in greater detail below, the trace cache generator may evict traces from trace cache 160. When a trace is evicted from the trace cache 160, there may be a fairly high probability that the instructions included in the evicted trace will be re-executed in the near term. In anticipation of these instructions being needed again, upon evicting a trace, the trace generator 170 may cause prefetch unit 108 to fetch instructions corresponding to the evicted trace into instruction cache 106.

[0035] Trace cache 160 may include several trace cache entries. Each trace cache entry may store a group of instructions referred to as a trace. FIG. 2 illustrates one embodiment of a trace cache entry 162. In the embodiment of FIG. 2, a trace cache entry 162 may store up to eight decoded and/or partially decoded instructions in operation storage units 166A-166H. Throughout this disclosure, drawing elements identified by the same reference numeral and a unique alphabetic identifier (e.g., operation storage units 166A-166H) may be collectively referred to by that reference numeral alone (e.g., operation storage units 166). Note that other embodiments may include fewer or additional operation storage units 166, allowing storage of different numbers of operations within each trace cache entry 162.

[0036] Certain operation storage units 166 within a trace cache entry 162 may be reserved for certain types of operations. For example, in one embodiment, operation storage units 166A-166D may be used to store memory operations. Similarly, operation storage units 166E-166H may be used to store data operations. Note that other embodiments may associate certain types of operations with certain operation storage units differently (or not at all).

[0037] In addition to including several operation storage units 166, each trace cache entry 162 also includes an identifying tag 164 and flow control information 168. Tag 164 may be similar to a tag in instruction cache 106, allowing prefetch unit 108 to determine whether a given operation hits or misses in trace cache 160. For example, tag 164 may

include all or some of the address bits identifying an operation within the trace cache entry (e.g., the tag may include the address of the earliest operation, in program order, stored within that trace). In some embodiments, the tag may include enough information that operations may be independently addressable within a trace. In other embodiments, only the first operation within a trace may be addressable.

[0038] In the illustrated embodiment, each trace may include up to two branch operations. Other embodiments may include fewer or additional branch operations within each trace. Flow control information 168 may include a label for each branch operation included within the trace. The label may be an indication identifying the address to which control should branch depending on the resolution (taken, not taken) of a respective branch. Thus, each item of flow control information 168 may be associated with a particular branch operation. For example, in one embodiment, one flow control information storage location within a trace may be associated with the first branch operation in the trace and the other flow control information storage location may be associated with the second branch in the trace. Alternatively, the flow control information may include tags or other information identifying the branch operation with which that flow control information is associated. In yet other embodiments, a branch prediction and/or information identifying which flow control information corresponds to a branch operation may be stored with that branch operation within operation storage 166.

[0039] After trace generator 170 has been in operation for some time, many entries in trace cache 160 may contain valid traces. When trace generator 170 produces the next trace, there may not be an available trace cache entry in which the trace may be stored. Therefore, the trace generator may evict one of the entries in trace cache 160 to make room for the new trace. Trace generator 170 is an example of a means for identifying traces to be evicted from trace cache 160 and evicting the identified traces.

[0040] Each time a trace is evicted from the trace cache 160, the prefetch unit 108 may fetch corresponding instructions into the instruction cache 106. Thus, if the

instructions included in the evicted trace are again executed, those instructions may be re-fetched from the instruction cache (as opposed to having to be re-fetched from system memory 200). In one embodiment, the prefetch unit 108 may use the tag 164 and/or flow information 168 stored in the evicted trace to identify one or more lines of instructions to fetch into the instruction cache 106 in response to the trace cache eviction. For example, if the tag 164 of the evicted trace identifies the address of the first instruction in program order stored within the trace, the prefetch unit 108 may fetch a line of instructions including the instruction identified by the tag 164 into the instruction cache 106. If the evicted trace includes a predicted taken branch, the line of instructions fetched into instruction cache based on the tag 164 may include instructions that precede the branch in program order. Note that the line of instructions fetched into instruction cache 106 may not include all of the instructions that are included in the evicted trace.

[0041] In some embodiments, before a line is fetched into instruction cache 106 in response to a trace cache eviction, the instruction cache 106 may be searched to determine if the line storing the instructions is resident in the instruction cache. If the line storing the instructions is not resident in the instruction cache, a prefetch cycle may be initiated to fetch the required line into instruction cache from level two cache or system memory. Alternatively, the fetch from level two cache or system memory 200 may be initiated without checking to see if the line is present in the instruction cache 106.

[0042] In some embodiments, the prefetch unit 108 may fetch multiple lines of instructions into the instruction cache 106 in response to a trace cache eviction. For example, the prefetch unit 108 may use the tag 164 of the evicted trace to select one line of instructions to fetch into the instruction cache 106. In some embodiments, the flow control information 168 for the evicted trace may include enough information for the prefetch unit 108 to identify lines of instructions that include the instructions following predicted-taken branches within the trace. Thus, the prefetch unit 108 may use the flow control information 168 for the evicted trace to identify one or more additional lines to fetch into the instruction cache 106 in response to the trace cache 160 eviction. In some embodiments, the prefetch unit 108 may be configured to fetch a number of lines that is

proportional to the number of branch operations included in the evicted trace (e.g., as indicated by flow control information 168).

5 **[0043]** In other embodiments, each fetch to instruction cache 106 in response to a trace cache 160 eviction may include two lines. For example, if the instruction whose address corresponds to the tag 164 of an evicted trace is stored near the end of a line of instructions, instructions included in the trace may correspond to instructions stored in the next line of the instruction cache 106. The prefetch unit 108 may fetch both lines to increase the likelihood that more of the instructions in the evicted trace will be brought
10 into the instruction cache.

[0044] In some embodiments, an evicted trace may not contain many instructions relative to the number of instructions that could have been stored in that trace. In such situations, the prefetch unit 108 may be configured to prefetch fewer lines into the
15 instruction cache 106 than would be prefetched if the evicted trace contained a relatively large number of instructions.

[0045] In certain cases, analysis of the evicted trace may indicate that the probability that the instructions corresponding to the trace will be re-executed near term is quite
20 small. For example, trace generator 170 may keep a record of the number of times each trace is used. During the process of selecting of a trace victim, the trace generator may determine that a trace has not been used since it entered the trace cache and therefore, it may be selected for eviction. In such instances, the trace generator may inhibit the fetching into the instruction cache of the group of instructions corresponding
25 to the evicted trace.

[0046] In some embodiments, the trace cache 160 may be implemented in multiple tiers. Such an implementation may be analogous to the level one and level two caches that store instructions and data within the microprocessor. For example, a trace for which
30 there is no longer room in the level one trace cache may be stored in the level two trace cache. The prefetch unit 108 may be configured to prefetch lines into instruction cache

106 based on trace cache evictions from the highest level (e.g., level two in a two-level trace cache) but not to prefetch lines into instruction cache 106 based on trace cache evictions from lower levels (e.g., level one) of the trace cache 160.

5 **[0047]** FIG. 3 is a flowchart of a method for operating a microprocessor containing a trace cache, according to one embodiment. As shown at 301, a trace is generated requiring the allocation of an entry in trace cache. A trace may be generated from instructions that have been executed and retired in some embodiments. In other embodiments traces may be generated from instructions that are awaiting execution.
10 Block 303 illustrates that if there is no unused entry available in the trace cache, then a trace currently resident in the trace cache may be selected for eviction. In some embodiments, the eviction process may propagate through successive levels of trace cache until a trace is evicted from the highest level of trace cache. Traces may be selected for eviction according to a variety of criteria including least recently used and/or
15 least frequently used.

[0048] The evicted trace may be analyzed as shown at block 305 in order to determine the line(s) of instructions to which it corresponds. The analysis may depend on the tag of the first instruction in program order within the trace, how fully populated
20 the trace is, and/or whether any branch instructions within the trace are predicted taken, among other things. Based on this analysis, one or more lines of instructions corresponding to the evicted trace may be fetched into the instruction cache at 309 from level two cache, system memory, or other locations.

25 Exemplary Computer Systems

[0049] FIG.4 shows a block diagram of one embodiment of a computer system 400 that includes a microprocessor 100 coupled to a variety of system components through a bus bridge 402. Microprocessor 100 may include an embodiment of a dispatch unit 104, a trace cache 160, a trace cache generator 170, and instruction cache 106, and/or a
30 prefetch unit 108 as described above. Other embodiments of a computer system are possible and contemplated.

[0050] In the depicted system, a main memory 200 is coupled to bus bridge 402 through a memory bus 406, and a graphics controller 408 is coupled to bus bridge 402 through an AGP bus 410. Several PCI devices 412A-412B are coupled to bus bridge 402 through a PCI bus 414. A secondary bus bridge 416 may also be provided to accommodate an electrical interface to one or more EISA or ISA devices 418 through an EISA/ISA bus 420. In this example, microprocessor 100 is coupled to bus bridge 402 through a CPU bus 424 and to an optional L2 cache 428. In some embodiments, the microprocessor 100 may include an integrated L1 cache (not shown).

10

[0051] Bus bridge 402 provides an interface between microprocessor 100, main memory 404, graphics controller 408, and devices attached to PCI bus 414. When an operation is received from one of the devices connected to bus bridge 402, bus bridge 402 identifies the target of the operation (e.g., a particular device or, in the case of PCI bus 414, that the target is on PCI bus 414). Bus bridge 402 routes the operation to the targeted device. Bus bridge 402 generally translates an operation from the protocol used by the source device or bus to the protocol used by the target device or bus.

15

[0052] In addition to providing an interface to an ISA/EISA bus for PCI bus 414, secondary bus bridge 416 may incorporate additional functionality. An input/output controller (not shown), either external from or integrated with secondary bus bridge 416, may also be included within computer system 400 to provide operational support for a keyboard and mouse 422 and for various serial and parallel ports. An external cache unit (not shown) may also be coupled to CPU bus 424 between microprocessor 100 and bus bridge 402 in other embodiments. Alternatively, the external cache may be coupled to bus bridge 402 and cache control logic for the external cache may be integrated into bus bridge 402. L2 cache 428 is shown in a backside configuration to microprocessor 100. It is noted that L2 cache 428 may be separate from microprocessor 100, integrated into a cartridge (e.g., slot 1 or slot A) with microprocessor 100, or even integrated onto a semiconductor substrate with microprocessor 100.

20

25

30

[0053] Main memory 200 is a memory in which application programs are stored and from which microprocessor 100 primarily executes. A suitable main memory 200 may include DRAM (Dynamic Random Access Memory). For example, a plurality of banks of SDRAM (Synchronous DRAM) or Rambus DRAM (RDRAM) may be suitable.

5

[0054] PCI devices 412A-412B are illustrative of a variety of peripheral devices such as network interface cards, video accelerators, audio cards, hard or floppy disk drives or drive controllers, SCSI (Small Computer Systems Interface) adapters and telephony cards. Similarly, ISA device 418 is illustrative of various types of peripheral devices, such as a modem, a sound card, and a variety of data acquisition cards such as GPIB or field bus interface cards.

10

[0055] Graphics controller 408 is provided to control the rendering of text and images on a display 426. Graphics controller 408 may embody a typical graphics accelerator generally known in the art to render three-dimensional data structures that can be effectively shifted into and from main memory 200. Graphics controller 408 may therefore be a master of AGP bus 410 in that it can request and receive access to a target interface within bus bridge 402 to thereby obtain access to main memory 200. A dedicated graphics bus accommodates rapid retrieval of data from main memory 404. For certain operations, graphics controller 408 may further be configured to generate PCI protocol transactions on AGP bus 410. The AGP interface of bus bridge 402 may thus include functionality to support both AGP protocol transactions as well as PCI protocol target and initiator transactions. Display 426 is any electronic display upon which an image or text can be presented. A suitable display 426 includes a cathode ray tube ("CRT"), a liquid crystal display ("LCD"), etc.

15

20

25

[0056] It is noted that, while the AGP, PCI, and ISA or EISA buses have been used as examples in the above description, any bus architectures may be substituted as desired. It is further noted that computer system 400 may be a multiprocessing computer system including additional microprocessors (e.g., microprocessor 100a shown as an optional component of computer system 400). Microprocessor 100a may be similar to

30

microprocessor 100. More particularly, microprocessor 100a may be an identical copy of microprocessor 100 in one embodiment. Microprocessor 100a may be connected to bus bridge 402 via an independent bus (as shown in FIG.4) or may share CPU bus 224 with microprocessor 100. Furthermore, microprocessor 100a may be coupled to an optional
5 L2 cache 428a similar to L2 cache 428.

[0057] Turning now to FIG.5, another embodiment of a computer system 400 that may include a microprocessor 100 that includes a dispatch unit 104, a trace cache 160, and/or a trace cache generator 170 as described above is shown. Other embodiments are
10 possible and contemplated. In the embodiment of FIG.5, computer system 400 includes several processing nodes 612A, 612B, 612C, and 612D. Each processing node is coupled to a respective memory 614A-614D via a memory controller 616A-616D included within each respective processing node 612A-612D. Additionally, processing nodes 612A-
15 612D include interface logic used to communicate between the processing nodes 612A-612D. For example, processing node 612A includes interface logic 618A for communicating with processing node 612B, interface logic 618B for communicating with processing node 612C, and a third interface logic 618C for communicating with yet another processing node (not shown). Similarly, processing node 612B includes interface
20 logic 618D, 618E, and 618F; processing node 612C includes interface logic 618G, 618H, and 618I; and processing node 612D includes interface logic 618J, 618K, and 618L. Processing node 612D is coupled to communicate with a plurality of input/output devices (e.g., devices 620A-620B in a daisy chain configuration) via interface logic 618L. Other processing nodes may communicate with other I/O devices in a similar fashion.

25 [0058] Processing nodes 612A-612D implement a packet-based link for inter-processing node communication. In the present embodiment, the link is implemented as sets of unidirectional lines (e.g., lines 624A are used to transmit packets from processing node 612A to processing node 612B and lines 624B are used to transmit packets from processing node 612B to processing node 612A). Other sets of lines 624C-624H are used
30 to transmit packets between other processing nodes as illustrated in FIG.5. Generally, each set of lines 624 may include one or more data lines, one or more clock lines

corresponding to the data lines, and one or more control lines indicating the type of packet being conveyed. The link may be operated in a cache coherent fashion for communication between processing nodes or in a non-coherent fashion for communication between a processing node and an I/O device (or a bus bridge to an I/O bus of conventional construction such as the PCI bus or ISA bus). Furthermore, the link may be operated in a non-coherent fashion using a daisy-chain structure between I/O devices as shown. It is noted that a packet to be transmitted from one processing node to another may pass through one or more intermediate nodes. For example, a packet transmitted by processing node 612A to processing node 612D may pass through either processing node 612B or processing node 612C as shown in FIG.5. Any suitable routing algorithm may be used. Other embodiments of computer system 400 may include more or fewer processing nodes than the embodiment shown in FIG.5.

[0059] Generally, the packets may be transmitted as one or more bit times on the lines 624 between nodes. A bit time may be the rising or falling edge of the clock signal on the corresponding clock lines. The packets may include command packets for initiating transactions, probe packets for maintaining cache coherency, and response packets from responding to probes and commands.

[0060] Processing nodes 612A-612D, in addition to a memory controller and interface logic, may include one or more microprocessors. Broadly speaking, a processing node includes at least one microprocessor and may optionally include a memory controller for communicating with a memory and other logic as desired. More particularly, each processing node 612A-612D may include one or more copies of microprocessor 100. External interface unit 18 may include the interface logic 618 within the node, as well as the memory controller 616.

[0061] Memories 614A-614D may include any suitable memory devices. For example, a memory 614A-614D may include one or more RAMBUS DRAMs (RDRAMs), synchronous DRAMs (SDRAMs), static RAM, etc. The address space of computer system 400 is divided among memories 614A-614D. Each processing node

612A-612D may include a memory map used to determine which addresses are mapped to which memories 614A-614D, and hence to which processing node 612A-612D a memory request for a particular address should be routed. In one embodiment, the coherency point for an address within computer system 400 is the memory controller 5 616A-616D coupled to the memory storing bytes corresponding to the address. In other words, the memory controller 616A-616D is responsible for ensuring that each memory access to the corresponding memory 614A-614D occurs in a cache coherent fashion. Memory controllers 616A-616D may include control circuitry for interfacing to memories 614A-614D. Additionally, memory controllers 616A-616D may include 10 request queues for queuing memory requests.

[0062] Interface logic 618A-618L may include a variety of buffers for receiving packets from the link and for buffering packets to be transmitted upon the link. Computer system 400 may employ any suitable flow control mechanism for transmitting 15 packets. For example, in one embodiment, each interface logic 618 stores a count of the number of each type of buffer within the receiver at the other end of the link to which that interface logic is connected. The interface logic does not transmit a packet unless the receiving interface logic has a free buffer to store the packet. As a receiving buffer is freed by routing a packet onward, the receiving interface logic transmits a message to the 20 sending interface logic to indicate that the buffer has been freed. Such a mechanism may be referred to as a "coupon-based" system.

[0063] I/O devices 620A-620B may be any suitable I/O devices. For example, I/O devices 620A-620B may include devices for communicate with another computer system 25 to which the devices may be coupled (e.g., network interface cards or modems). Furthermore, I/O devices 620A-620B may include video accelerators, audio cards, hard or floppy disk drives or drive controllers, SCSI (Small Computer Systems Interface) adapters and telephony cards, sound cards, and a variety of data acquisition cards such as GPIB or field bus interface cards. It is noted that the term "I/O device" and the term 30 "peripheral device" are intended to be synonymous herein.

[0064] As used herein, the terms “clock cycle” or “cycle” refer to an interval of time in which the various stages of the instruction processing pipelines complete their tasks. Instructions and computed values are captured by memory elements (such as registers or arrays) according to a clock signal defining the clock cycle. For example, a memory
5 element may capture a value according to the rising or falling edge of the clock signal.

[0065] Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

10